# Practical 14

## AIM: - Write a code using RAW sockets to implement packet sniffing.

### ALGORITHM:

- Start the program with root privileges.
- Create a raw socket bound to the link layer to receive all packets.
- Continuously receive raw frames from the socket.
- Parse the Ethernet header to get the EtherType.
- If EtherType indicates IPv4, parse the IP header to get source IP, destination IP, protocol and header length.
- Based on IP protocol field, parse TCP or UDP headers to obtain source/destination ports and header lengths.
- Extract a small payload preview (hex/ASCII) and print timestamp, src/dst IP & ports, protocol, packet length and payload preview.
- Repeat until the user stops the program (Ctrl+C).
- Close the socket and exit.

## CODE:

```
import socket
import struct
import binascii
import time

s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0003))
print("Packet sniffer started (run as root). Press Ctrl+C to stop.")
try:
  while True:
    raw_data, addr = s.recvfrom(65535)
    ts = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    eth_header = struct.unpack('!6s6sH', raw_data[:14])
    eth_proto = socket.ntohs(eth_header[2])
    if eth_proto != 0x0800:
      continue
    ip_start = 14
    ip_header = struct.unpack('!BBHHHBBH4s4s', raw_data[ip_start:ip_start+20])
    version_ihl = ip_header[0]
    ihl = version_ihl & 0x0F
    ip_header_len = ihl * 4
    total_length = ip_header[2]
    proto = ip_header[6]
    src_ip = socket.inet_ntoa(ip_header[8])
    dst_ip = socket.inet_ntoa(ip_header[9])
    transport_start = ip_start + ip_header_len
```

# Practical 14

```
        proto_name = ""
        src_port = ""
        dst_port = ""
        payload = b""
        if proto == 6:
            proto_name = "TCP"
            tcp_header = raw_data[transport_start:transport_start+20]
            if len(tcp_header) >= 20:
                t = struct.unpack('!HHLLBBHHH', tcp_header)
                src_port = t[0]
                dst_port = t[1]
                data_offset = (t[4] >> 4) * 4
                payload = raw_data[transport_start + data_offset:transport_start + total_length - ip_header_len + ip_start]
        elif proto == 17:
            proto_name = "UDP"
            udp_header = raw_data[transport_start:transport_start+8]
            if len(udp_header) >= 8:
                u = struct.unpack('!HHHH', udp_header)
                src_port = u[0]
                dst_port = u[1]
                payload = raw_data[transport_start + 8:transport_start + total_length - ip_header_len + ip_start]
        else:
            proto_name = f"IP_PROTO_{proto}"
            payload = raw_data[transport_start:transport_start + total_length - ip_header_len + ip_start]
        payload_preview = binascii.hexlify(payload[:32]).decode() if payload else ""
        ascii_preview = ''.join((chr(b) if 32 <= b <= 126 else '.') for b in payload[:32])
        print(f"[{ts}] {proto_name} {src_ip}:{src_port} -> {dst_ip}:{dst_port} len={total_length}
preview_hex={payload_preview} preview_ascii={ascii_preview}")
except KeyboardInterrupt:
    print("\nStopping sniffer.")
finally:
    s.close()
```

**Input:**

```
sudo python3 packet_sniffer.py
```

**Output:**

```
Packet sniffer started (run as root). Press Ctrl+C to stop.
[2025-11-04 14:12:03] TCP 192.168.1.10:38402 -> 142.250.190.78:443 len=60
preview_hex=17030301240100012003 03... preview_ascii=..   ...
[2025-11-04 14:12:03] TCP 142.250.190.78:443 -> 192.168.1.10:38402 len=60
preview_hex=1703030034010000300303... preview_ascii=..   ..
[2025-11-04 14:12:05] ICMP 192.168.1.10: -> 8.8.8.8: len=84 preview_hex=0800f7ff00017b4f00010000...
preview_ascii=....{O....
[2025-11-04 14:12:08] UDP 192.168.1.10:5353 -> 224.0.0.251:5353 len=52
preview_hex=00042e6d61726b2d056d79686f... preview_ascii=...mark-.myho
Stopping sniffer.
```